

## Capítulo TRINTA

### Localização (Localization)

#### Objetivos do Exame

- Ler e definir a localidade usando o objeto `Locale`.
- Criar e ler um arquivo de propriedades (Properties).
- Construir um pacote de recursos para cada localidade e carregar um pacote de recursos em uma aplicação.

---

#### Localização

Localização (abreviada como **l10n** por causa do número de caracteres entre a primeira e a última letra) é o mecanismo pelo qual uma aplicação é adaptada para um idioma e região específicos.

Está relacionada ao conceito de **internacionalização** (abreviada como **i18n** pela mesma razão que localização), que trata do design de uma aplicação capaz de lidar com diferentes idiomas e regiões.

As coisas mais comuns que podem ser personalizadas por idioma e/ou região são **mensagens, datas e números**.

Em Java, tudo começa com uma classe: `java.util.Locale`.

A classe `Locale` basicamente representa um idioma e um país embora, para ser preciso, uma localidade pode conter as seguintes informações:

- Um código de idioma ISO 639 alpha-2 ou alpha-3, como `ja` (Japonês)
- Um código de país ISO 3166 alpha-2 ou código de área numérico-3 UN M.49, como `JP` (Japão)
- Um nome de variante, geralmente vazio, mas pode ser qualquer string
- Um código de script ISO 15924 alpha-4, como `Latn` (Latim)
- Um conjunto de extensões representadas por caracteres únicos, como `u`.

Mas na maioria das vezes, apenas trabalhamos com **idiomas e países**.

---

#### Uma representação de `Locale`

- A parte do idioma é **obrigatória**
- A parte do país é **opcional**

---



#### Exemplo: `fr_CA`

- Observe as **letras minúsculas** na parte do idioma
- Em seguida, o **sublinhado** para separação
- Observe as **letras maiúsculas** na parte do país

---

Você pode obter a localidade padrão da sua máquina com:

```
java
Locale locale = Locale.getDefault();
```

 Copiar  Editar

E obter informações como:

```
java Copiar Editar

System.out.println("Country Code: " + locale.getCountry());
System.out.println("Country Name: " + locale.getDisplayCountry());
System.out.println("Language Code: " + locale.getLanguage());
System.out.println("Language Name: " + locale.getDisplayLanguage());
```

Saída (observe como os nomes reais são localizados em espanhol):

```
yaml Copiar Editar

Country Code: MX
Country Name: México
Language Code: es
Language Name: español
```

Você também pode obter todas as localidades suportadas pelo Java:

```
java Copiar Editar

Locale[] locales = Locale.getAvailableLocales();
Arrays.stream(locales)
    .forEach(System.out::println);
```

Isso imprimirá cerca de 160 localidades no formato idioma[\_país], por exemplo:

```
nginx Copiar Editar

it
pt_BR
ro_RO
```

---

## Definindo a localidade

Há três formas diferentes de criar uma instância de Locale:

### 1. Usando um construtor

Existem três construtores:

```
java Copiar Editar

Locale(String language)
Locale(String language, String country)
Locale(String language, String country, String variant)
```

Exemplo:

```
java Copiar Editar

Locale chinese = new Locale("zh");
Locale CHINA = new Locale("zh", "CN");
```

---

### 2. Usando o método de fábrica forLanguageTag(String)

Este método espera um código de idioma, por exemplo:

```
java
Locale german = Locale.forLanguageTag("de");
```

---

### 3. Usando Locale.Builder

Você pode definir as propriedades necessárias e construir o objeto no final, por exemplo:

```
java
Locale japan = new Locale.Builder()
    .setRegion("JP")
    .setLanguage("jp")
    .build();
```

Passar um argumento inválido para qualquer um dos três métodos acima **não lançará uma exceção**, apenas criará um objeto com opções inválidas que farão seu programa se comportar incorretamente:

```
java
Locale badLocale = new Locale("a", "A"); // Sem erro
System.out.println(badLocale); // Imprime a_A
```

---

A classe Locale também fornece **constantes predefinidas** para alguns idiomas e países comuns, por exemplo:

- Locale.GERMAN
- Locale.KOREAN
- Locale.UK
- Locale.ITALY

Para o exame, você **não precisa conhecer todas essas constantes**, ou códigos de idioma e país obscuros, apenas que há quatro formas de começar a trabalhar com localidades.

---

Uma vez que você tem um objeto Locale, pode mudar a localidade do seu programa com o método `setDefault(Locale)`:

```
java
System.out.println(Locale.getDefault()); // Suponha que imprima en_GB
Locale.setDefault(new Locale("en", "US"));
System.out.println(Locale.getDefault()); // Agora imprime en_US
```

---

### Arquivos de Propriedades

Arquivos de propriedades definem strings em pares **chave/valor** separados por linhas.

Há algumas regras, como:

- Espaços no início da linha (se houver) são ignorados.
- Qualquer linha que comece com # ou ! será tratada como **comentário**.
- Você pode quebrar uma linha para fins de legibilidade com uma **barra invertida \**.

Exemplo de arquivo:

```
ini
# Mensagens relacionadas a vídeo
video.added = The video has been added
video.deleted = The video has been deleted
```

Para lê-lo, crie uma instância de `java.util.Properties` e carregue-a com um `java.io.Reader` ou `java.io.InputStream`, por exemplo:

```
java
Properties prop = new Properties();
try (InputStream is = getClassLoader()
    .getResourceAsStream("Messages.properties")) {
    prop.load(is); // Carrega propriedades
    System.out.println(prop.getProperty("video.added")); // Imprime "The video has been added"
    System.out.println(prop.getProperty("video.add", "default")); // Valor padrão
    Enumeration<?> e = prop.propertyNames(); // Obtém todas as chaves
} catch (IOException e) {
    e.printStackTrace();
}
```

---

## Pacotes de Recursos (Resource Bundles)

Para localizar uma aplicação, temos os **Resource Bundles (Pacotes de Recursos)**, que definem um conjunto de chaves com valores localizados. Pacotes de recursos podem ser arquivos de propriedades ou classes.

Para dar suporte a isso, temos uma classe abstrata `java.util.ResourceBundle` com duas subclasses:

- `java.util.PropertyResourceBundle`  
Cada localidade é representada por um arquivo de propriedades. As chaves e valores são do tipo `String`.
- `java.util.ListResourceBundle`  
Cada localidade é representada por uma subclasse dessa classe que sobrescreve o método `Object[][] getContents()`. O array retornado representa as chaves e valores. As chaves devem ser do tipo `String`, mas os valores podem ser de qualquer objeto.

Em ambos os métodos, o nome (do arquivo ou da classe) segue uma convenção que permite ao Java procurar pacotes de recursos e associá-los às suas localidades correspondentes.

Essa convenção de nomes é:

```
go
package.Bundle_idioma_pais_variante
```

Por exemplo:

```
com.example.MyBundle_fr_FR
```

Apenas o nome do pacote de recurso é obrigatório (e o nome do pacote se não for o padrão).

Por exemplo, podemos ter pacotes com os seguintes nomes (assumindo que estamos trabalhando com arquivos de propriedades, embora seja o mesmo com classes):

matlab

Copiar Editar

```
MyBundle.properties  
MyBundle_en.properties  
MyBundle_en_NZ.properties  
MyBundle_en_US.properties
```

Para determinar qual pacote pertence a uma localidade particular, o Java tenta encontrar o pacote mais específico que corresponde às propriedades da localidade.

Isso significa que:

1. O Java primeiro procura um pacote cujo nome corresponda à localidade completa:  
package.bundle\_idioma\_pais\_variante
2. Se não encontrar um, ele remove o último componente do nome e repete a busca:  
package.bundle\_idioma\_pais
3. Se ainda não encontrar, novamente remove o último componente e repete a busca:  
package.bundle\_idioma
4. Se ainda não encontrar, o último componente é removido novamente, restando apenas o nome do pacote:  
package.bundle

Se nada for encontrado, uma `MissingBundleException` é lançada.

Se uma **classe** e um **arquivo de propriedades** compartilharem o mesmo nome, o Java dá **prioridade à classe**.

Mas há outro ponto importante.

No seu programa, você pode usar as chaves do pacote de recursos correspondente **e de QUALQUER UM de seus PAIS**.

Os pais de um pacote de recursos são os com o mesmo nome, mas com menos componentes. Por exemplo, os pais de `MyBundle_es_ES` são:

- `MyBundle_es`
- `MyBundle`

Por exemplo, vamos assumir a localidade padrão `en_US`, e que seu programa está usando esses e outros arquivos de propriedades, todos no pacote padrão, com os valores:

java

Copiar Editar

```
MyBundle_EN.properties  
s = buddy  
  
MyBundle_es_ES.properties  
s = tío  
  
MyBundle_es.properties  
s = amigo  
  
MyBundle.properties  
hi = Hola
```

Podemos criar um pacote de recursos assim:

```

java Copiar Editar

public class Test {
    public static void main(String[] args) {
        Locale spain = new Locale("es", "ES");
        Locale spanish = new Locale("es");
        ResourceBundle rb = ResourceBundle.getBundle("MyBundle", spain);
        System.out.format("%s %s\n",
            rb.getString("hi"), rb.getString("s"));

        rb = ResourceBundle.getBundle("MyBundle", spanish);
        System.out.format("%s %s\n",
            rb.getString("hi"), rb.getString("s"));
    }
}

```

Saída:

```

nginx Copiar Editar

Hola tío
Hola amigo

```

Como você pode ver, cada localidade escolhe valores diferentes para a chave `s`, mas ambas usam o mesmo valor para `hi`, já que essa chave está definida em seu pai.

Se você **não especificar** uma localidade, a classe `ResourceBundle` usará a **localidade padrão do seu sistema**:

```

java Copiar Editar

ResourceBundle rb = ResourceBundle.getBundle("MyBundle");
System.out.format("%s %s\n",
    rb.getString("hi"), rb.getString("s"));

```

Como assumimos que a localidade padrão é `en_US`, a saída será:

```

nginx Copiar Editar

Hola buddy

```

Também podemos obter todas as chaves de um pacote de recursos com o método `keySet()`:

```

java Copiar Editar

ResourceBundle rb = ResourceBundle.getBundle("MyBundle", spain);
Set<String> keys = rb.keySet();
keys.stream()
    .forEach(key ->
        System.out.format("%s %s\n", key, rb.getString(key)));

```

Saída (observe que também imprime a chave do pai):

```

nginx Copiar Editar

hi Hola
s tío

```

Se ao invés de usar arquivos de propriedades estivermos usando classes, o programa seria assim:

```

java
Copiar Editar

package bundles;
public class MyBundle_EN extends ListResourceBundle {
    @Override
    protected Object[][] getContents() {
        return new Object[][] {
            { "s", "buddy" }
        };
    }
}

package bundles;
public class MyBundle_es_ES extends ListResourceBundle {
    @Override
    protected Object[][] getContents() {
        return new Object[][] {
            { "s", "tío" }
        };
    }
}

package bundles;
public class MyBundle_es extends ListResourceBundle {
    @Override
    protected Object[][] getContents() {
        return new Object[][] {
            { "s", "amigo" }
        };
    }
}

package bundles;
public class MyBundle extends ListResourceBundle {
    @Override
    protected Object[][] getContents() {
        return new Object[][] {
            { "hi", "Hola" }
        };
    }
}

```

```

java
Copiar Editar

public class Test {
    public static void main(String[] args) {
        Locale spain = new Locale("es", "ES");
        Locale spanish = new Locale("es");

        ResourceBundle rb =
            ResourceBundle.getBundle("bundles.MyBundle", spain);
        System.out.format("%s %s\n",
            rb.getString("hi"), rb.getString("s"));

        rb = ResourceBundle.getBundle("bundles.MyBundle", spanish);
        System.out.format("%s %s\n",
            rb.getString("hi"), rb.getString("s"));
    }
}

```

A única coisa que mudou na classe Test foi o nome do pacote (precisamos referenciar o pacote). Isso não deveria surpreender você, afinal, tanto PropertyResourceBundle quanto ListResourceBundle herdam da mesma classe.

Lembre-se também, ao usar classes podemos ter valores de tipos diferentes de String, por exemplo:

```
java Copiar Editar

public class MyBundle extends ListResourceBundle {
    @Override
    protected Object[][] getContents() {
        return new Object[][] {
            { "hi", "Hola" },
            { "number", new Integer(100) }
        };
    }
}
```

Para obter um valor de objeto, usamos:

```
java Copiar Editar

Integer num = (Integer) rb.getObject("number");
```

Ao invés de `rb.getString(key)`. De fato, este método é apenas um atalho para:

```
java Copiar Editar

String val = (String) rb.getObject("hi");
```

---

## Pontos-chave

- **Localização** (abreviada como **L10N** por causa do número de caracteres entre a primeira e a última letra) é o mecanismo pelo qual uma aplicação é adaptada a um idioma e região específicos.
- A classe `java.util.Locale` basicamente representa um idioma e um país, e é o ponto de partida para a localização em Java.
- Você pode obter a localidade padrão da sua máquina com:

```
java Copiar Editar

Locale locale = Locale.getDefault();
```

Você também pode obter todas as localidades suportadas pelo Java:

```
java Copiar Editar

Locale[] locales = Locale.getAvailableLocales();
```

Você pode criar um objeto `Locale` usando um construtor:

```
java Copiar Editar

Locale(String language)
Locale(String language, String country)
Locale(String language, String country, String variant)
```

Usando o método de fábrica `forLanguageTag(String)`:

```
java Copiar Editar

Locale german = Locale.forLanguageTag("de");
```



Usando Locale.Builder:

```
java Copiar Editar

Locale japan = new Locale.Builder()
    .setRegion("JP")
    .setLanguage("jp")
    .build();
```

Usando constantes predefinidas para alguns idiomas e países comuns, por exemplo:

```
java Copiar Editar

Locale.GERMAN
Locale.KOREAN
```

Uma vez que você tenha um objeto Locale, pode mudar a localidade do seu programa com o método `setDefault(Locale)`:

```
java Copiar Editar

Locale.setDefault(new Locale("en", "US"));
```

- ❑ **Arquivos de propriedades** definem strings em pares **chave/valor** separados por linhas.
  - ❑ Para localizar uma aplicação, temos **Resource Bundles**, que definem um conjunto de chaves com valores localizados. Resource Bundles podem ser **arquivos de propriedades** ou **classes**.
  - ❑ `java.util.PropertyResourceBundle`: cada localidade é representada por um arquivo de propriedades. Chaves e valores são do tipo `String`.
  - ❑ `java.util.ListResourceBundle`: cada localidade é representada por uma subclasse que sobrescreve o método `Object[][] getContents()`. O array retornado representa as chaves e valores. Chaves devem ser do tipo `String`, mas os valores podem ser qualquer objeto.
  - ❑ Para determinar qual pacote pertence a uma localidade específica, o Java tenta encontrar o pacote mais específico que corresponda às propriedades da localidade.
  - ❑ Se não conseguir localizar um, o último componente do nome é removido até restar apenas o nome do pacote.
  - ❑ Se nada for encontrado, uma `MissingBundleException` é lançada.
  - ❑ Se uma classe e um arquivo de propriedades compartilharem o mesmo nome, o Java dá **prioridade à classe**.
  - ❑ Você pode usar as chaves do pacote de recursos correspondente e **QUALQUER UM de seus PAIS**.
  - ❑ Os pais de um pacote de recursos são os com o mesmo nome, mas com menos componentes.
-

## Autoavaliação

### 1. Dado:

```
java Copiar Editar

public class Question_30_1 {
    public static void main(String[] args) {
        Locale locale = new Locale("", "");
        ResourceBundle rb = ResourceBundle.getBundle("Bundle1", locale);
        System.out.println(rb.getString("key1"));
    }
}
```

Bundle1.properties

```
ini Copiar Editar

key1 = Hi
```

Qual é o resultado?

- A. Hi
  - B. null
  - C. A compilação falha
  - D. Uma exceção ocorre em tempo de execução
- 

### 2. Quais das seguintes são formas válidas de criar uma localidade?

- A. new Locale();
  - B. Locale.Builder().setLanguage("de");
  - C. new Locale.Builder().setRegion("DE").build();
  - D. Locale.forRegionTag("it");
- 

### 3. Assumindo uma localidade padrão de\_DE, qual dos seguintes pacotes de recursos será carregado primeiro com:

```
java Copiar Editar

ResourceBundle rb = new ResourceBundle("MyBundle");
```

- A. MyBundle.class
  - B. MyBundle.properties
  - C. MyBundle\_de.class
  - D. MyBundle\_de.properties
- 

### 4. Dado:

```
java Copiar Editar

public class Question_30_4 {
    public static void main(String[] args) {
        Locale locale = new Locale("en", "CA");
        System.out.println(rb.getString(locale));
    }
}
```

Qual é o resultado?

- A. en
  - B. en\_CA
  - C. CA
  - D. CA\_en
- 

**5. Quais das seguintes são formas válidas de obter um valor dada sua chave de um pacote de recursos do tipo arquivo de propriedades rb?**

- A. `rb.getValue("key");`
- B. `rb.getProperty("key");`
- C. `rb.getObject("key");`
- D. `rb.get("key");`